Priority Inversion and Blocking Tasks

Plus:

What other things do we need to learn to schedule our realtime systems but do not learn in this unit? A brief mention of some that will need to be considered.

This material has been adapted from chapter 6 slides provided with the textbook by Wolf “Computers as Components: principles of embedded system design” Morgan Kaufmann ©

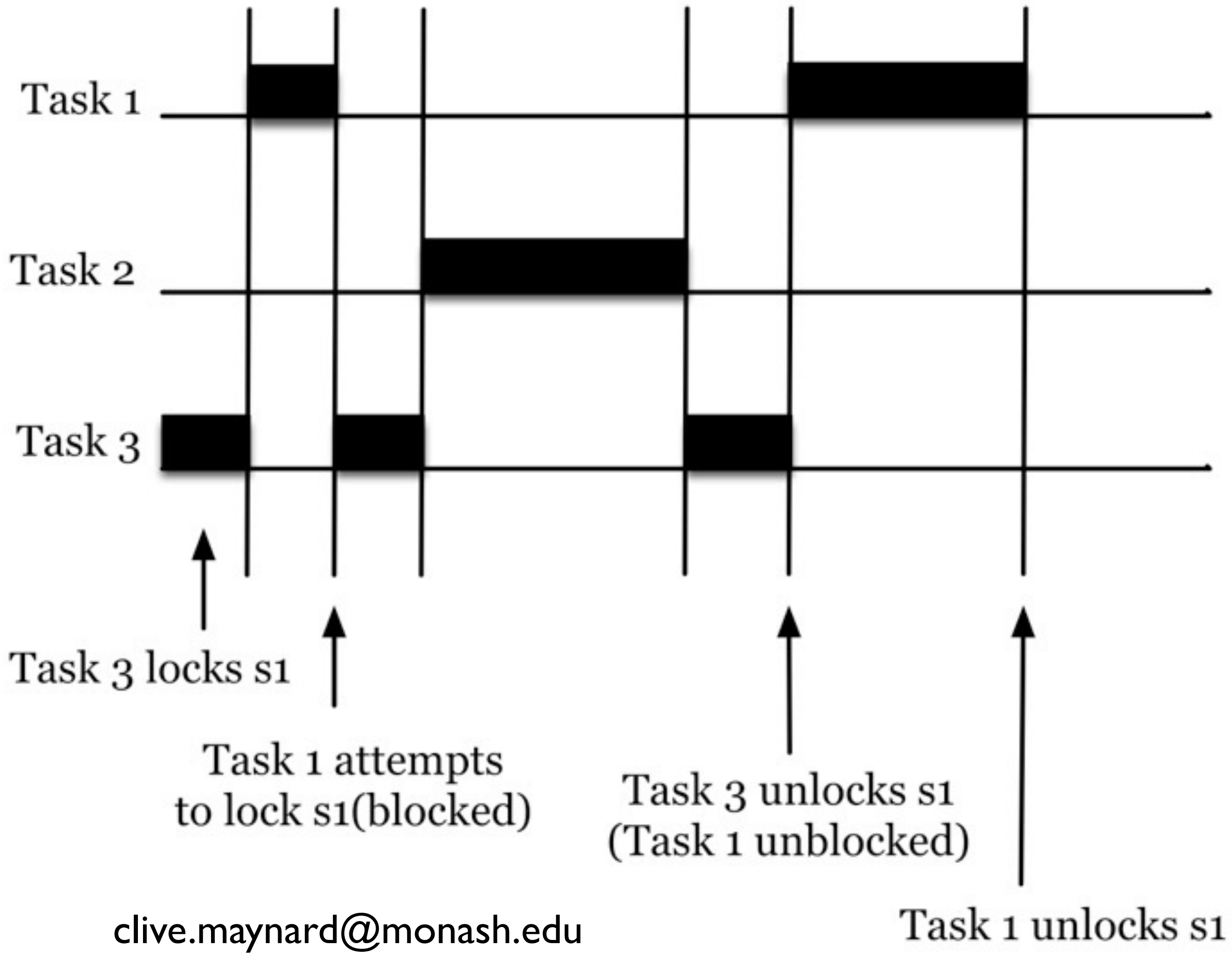
Priority inversion

- Priority inversion : low -priority process keeps high -priority process from running.
- Improper use of system resources can cause scheduling problems:
  - Low -priority process grabs I/O device.
  - High -priority device needs I/O device, but can’t get it until low -priority process is done.
- Can cause deadlock.

Revisiting the RMA Criteria

- The conditions under which the Liu and Layland (RMA) theorems were defined are more limited than is often experienced in practice.
- The first deviation from those conditions is to allow the tasks to require exclusive access to system resources. e.g. reading or writing values into memory locations. To prevent corruption of the data we only allow one task access to the resource at one time (a critical section). This simple change has considerable impact on the system and affects the schedulability analysis.

Unbounded Priority Inversion


Task 1
Task 2
Task 3
Task 3 locks s1
Task 1 attempts to lock s1(blocked)
Task 3 unlocks s1 (Task 1 unblocked)
Task 1 unlocks s1

Priority Inversion with Semaphores

- Three processes Task 1, Task 2, Task 3.
  - Highest priority Task 1 waits on semaphore that is to be signalled by Task 3.
  - Task 2 runs (& runs & runs …) since Task 1 is blocked.
- We have task 2 running preventing

task 3 and henceTask 1 from running.

- => priority inversion!

Unbounded Priority Inversion

- If this condition can arise then we cannot put a guarantee on task schedulability.
- A number of techniques to prevent unbounded priority inversion have been proposed but the best is
- **The Priority Ceiling Protocol.**

Priority Ceiling Protocol

- This has freedom from mutual deadlock
- And bounded priority inversion with, at most, one lower priority task able to block a higher priority task.

Priority Inheritance

- When a task T blocks the execution of a higher priority task then task T executes at the highest priority level of all the tasks blocked by T.

Solving priority inversion

- Give priorities to system resources.
- Have process inherit the priority of a resource

that it requests.

  - Low -priority process inherits priority of device if higher.

Execution Inheritance

- The protocol guarantees that a critical section is allowed to start execution only if the section will always execute at a priority level higher than the (inherited) priority levels of any preempted critical sections.

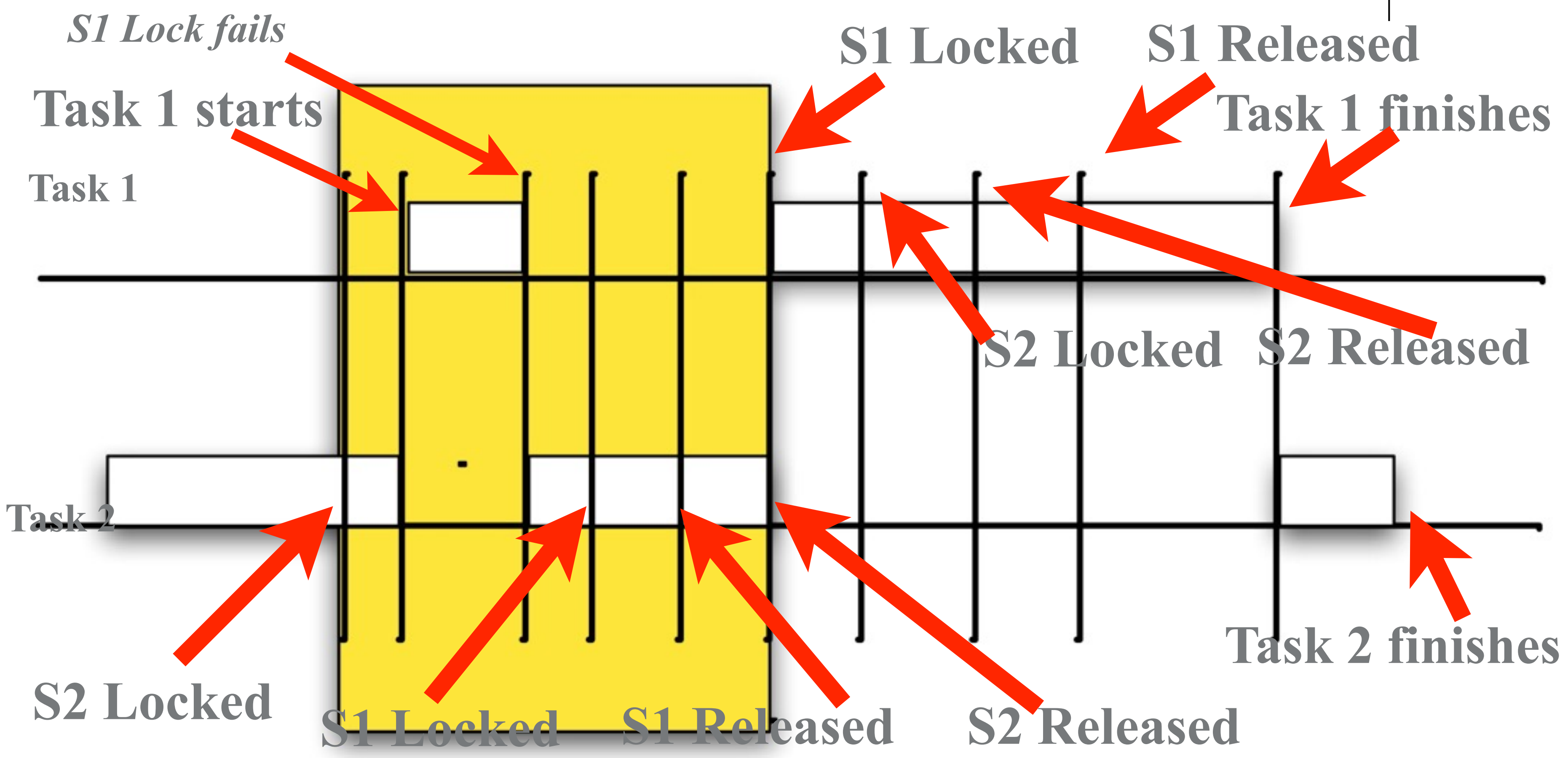
Contd…

- We define the priority ceiling of a semaphore (Locking process) S to be the highest priority of all tasks that may lock S.
- When a task T attempts to execute one of its critical sections, it will be suspended unless its priority is higher than the priority ceilings of all semaphores currently locked by tasks other than T.

Contd…

- If task T is unable to enter its critical section for this reason, the task that holds the lock on the semaphore with the highest priority is said to be blocking T, and hence, inherits the priority of T.
- As long as task T is not attempting to enter one of its critical sections, it will preempt every task that has a lower priority.

Task 1 requires P(S1)...P(S2)...V(S2)...V(S1)...
Task 2 requires P(S2)...P(S1)...V(S1)...V(S2)...



Notes:

- Both tasks use semaphores S1 and S2
- When Task 1 tries to lock S1 the OS process not only checks if S1 is locked but if any other task that locks S1 is already actively locking another semaphore.
- If so then Task 1 is not allowed to continue and the other task has its priority raised to the level of Task 1 for continued execution.

Blocking under the Priority Ceiling Protocol

- Both tasks use semaphores S1 and S2
- When Task 1 tries to lock S1 the OS process not only checks if S1 is locked but if any other task that locks S1 is already actively locking another semaphore.

RMA Theorem 3: (Blocking version of Theorem 1)

- A set of n periodic tasks using the priority ceiling protocol can be scheduled by the RMA for all task phasings, if the following condition is satisfied:

$$U_{total} = \sum_{i=1}^{n} \frac{C_i}{T_i}; B_{total} = \max_{i=1}^{n} \left(\frac{B_i}{T_i} \right); UB(n) = n \left(2^{\frac{1}{n}} - 1 \right)$$

$$U_{total} + B_{total} \le UB(n)$$

Example

Task #	C	T	B	U	Sum of U
1	4	40	0	0.1	0.1
2	10	150	8	0.067	0.167
3	20	180	0	0.111	0.278
4	10	250	5	0.04	0.318
5	80	300	0	0.267	0.584

Example contd…

- Calculate the total utilisation = 0.59
- U_{total} = 4/40 + 10/150 + etc
- Calculate the blocking term = 0.054
- B_{Total} =max(8/150,5/250) = 0.054
- Calculate the utilisation bound = 0.743
- Compare the effective utilisation and the bound 0.59 + 0.054 = 0.644 <0.743
- The task set is schedulable.

Pushthrough Blocking

- To apply the blocking version of Theorem 2 under the priority ceiling protocol you must remember that a response can only be blocked by at most a single critical section, therefore, the blocking term used is derived from the longest critical section of a lower priority task. This includes indirect blocking (also called Pushthrough Blocking).

Problem: (Bi is value caused by Taski)

Task #	C	T	B	U	Sum of U	B total (Max utilisation of lower tasks)
1	20	100	0	0.2	0.2	0.1
2	20	150	10	0.133	0.333	0.1
3	40	200	20	0.2	0.533	0.084
4	30	300	25	0.1	0.633	0.072
5	45	350	25	0.129	0.762	0.025= 10/400
6	20	400	10	0.05	0.767	

Problem contd…

- The first 4 tasks are schedulable including the blocking.
- How do we apply Theorem 2 to task 5?
- The total available time up to any scheduling point is decreased by the magnitude of the blocking that the task being checked may experience. In this case 10.
- Scheduling points for task 5 are:
  100, 150, 200, 300, 350

Theorem 2 applied to Task 5:

$$1x20+1x20+1x40+1x30+1x45 \le 100-10?$$

$$155 \le 90 Fail$$

$$2x20+2x20+1x40+1x30+1x45 \le 200-10?$$

$$195 \le 190 Fail$$

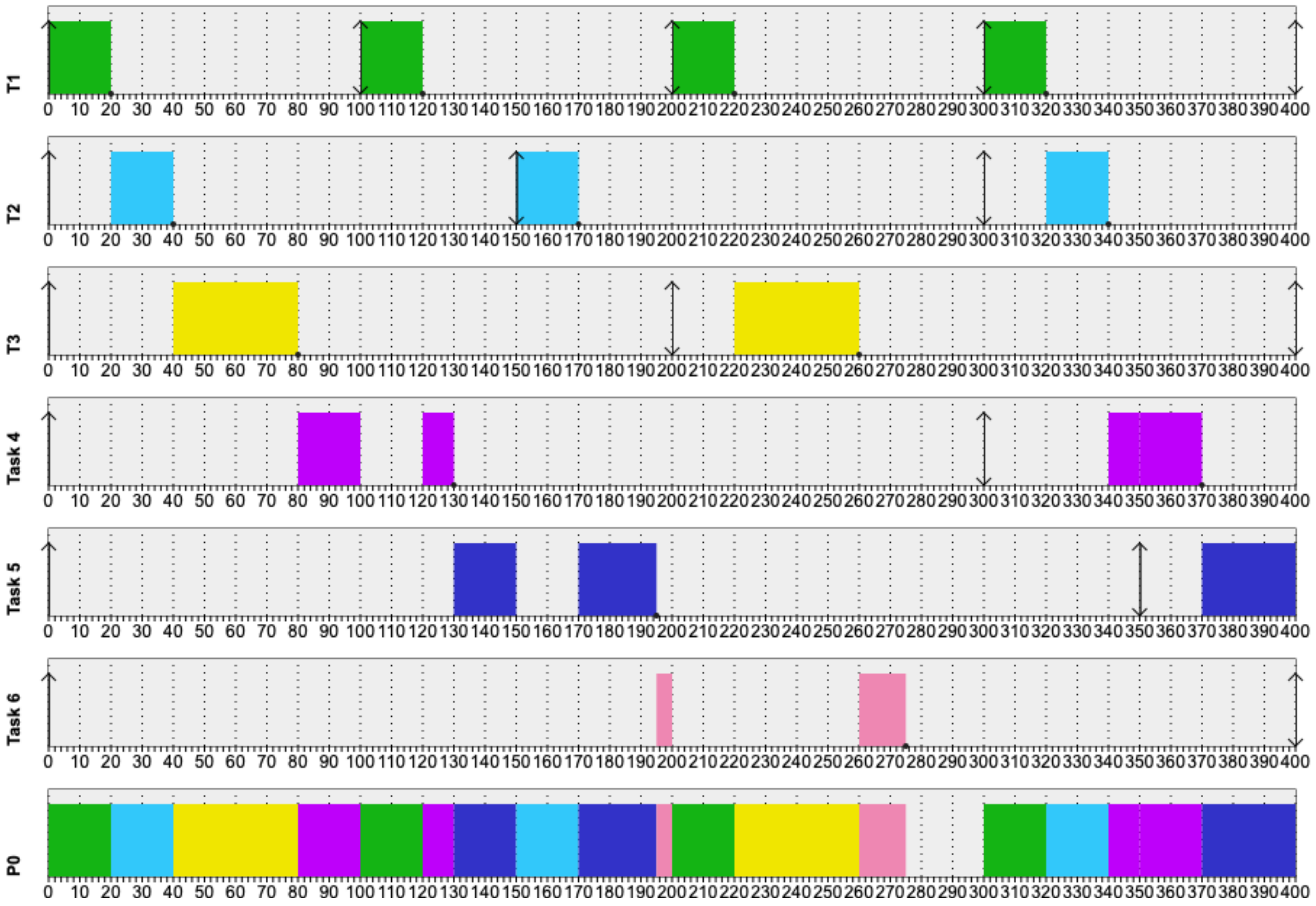
$$3x20+2x20+2x40+1x30+1x45 \le 300-10?$$

$$255 \le 290 OK$$

What about task 6?

Scheduling points are:
100, 150, 200, 300, 350, and 400
and Task 6 is the lowest priority task so there is no blocking component to consider.
Consider that Task 5 is schedulable by 255 time units and the next scheduling point is 300.
So there is enough time to complete Task 6 by 275 (20 time units)
Formally:
3x20+2x20+2x40+1x30+1x45+1x20 =275 <=300
The task set is schedulable.

What about task 6?



clive.maynard@monash.edu

Solution to Priority Inversion with semaphores

- Raise the priority of all processes occupying a resource (ie waited on semaphore but not yet signalled S) to the maximum of all processes blocked by the resource.
- Adopted by uC/OS - II with a mutex – a binary semaphore.
  - A task that holds a mutex token is raised in priority when another task is blocked waiting on the mutex.
  - The priority is restored when the task releases the mutex token.
  - See Example 8 on the unit website, reference manual for OSMutexCreate(), OSMutexPend() and OSMutexPost()

Mutual Exclusion and uCOSII

- Mutual exclusion semaphores can be used to reduce the priority inversion problem.
- To implement a classic mutex an RTOS needs to increase the priority of the lower priority task to the priority of the higher priority task until the lower priority task is done with the resource.
- **uCOSII does not allow multiple tasks of the same priority.**
- **This problem is removed by allocating a priority just above the highest priority task needing the mutex.** Designers have to be aware of this allocation requirement.

Future Studies or What else do we need to know?

- Aperiodic Tasks
  - How do we incorporate them into our scheduling predictions (Aperiodic/Sporadic Server)?
- Multiprocessors
  - How do we distribute tasks and scheduling activities between multiple processors?
- Cache and Realtime
  - What are the impacts of cache memories on scheduling and realtime requirements?